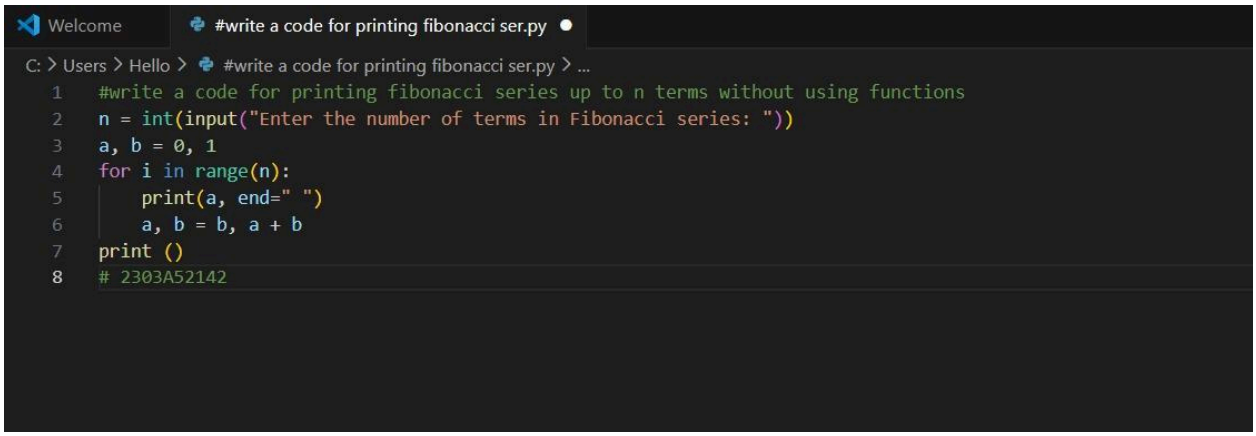
Assignment - 01
Baisa Vaishnavi
2303A52142

Task 1: AI-Generated Logic Without Modularization (Fibonacci Sequence Without Functions)

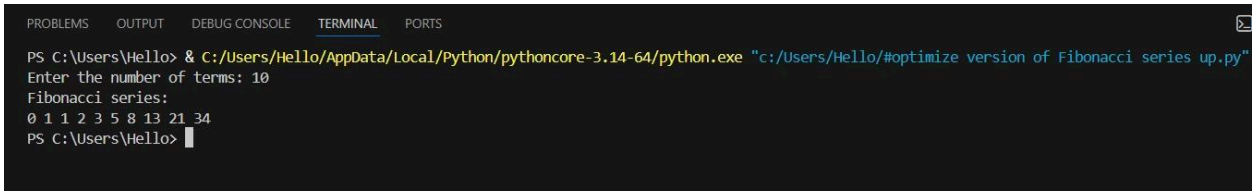
PROMPT:

Write a code for printing fibonacci series up to n terms

CODE:



Output:



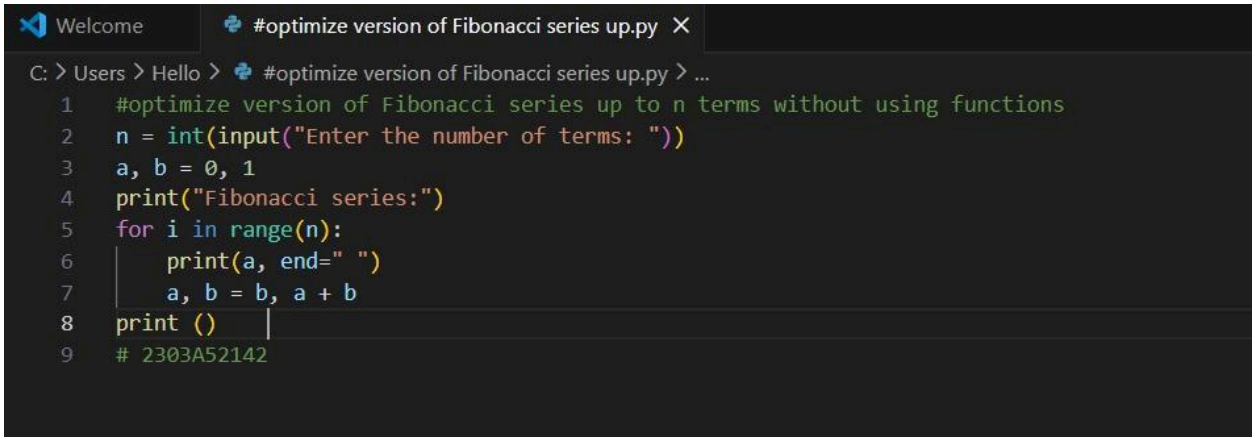
JUSTIFICATION

By completing this task, we have printed the required Fibonacci series up to n terms without using modularization. The logic is written directly in the main program without defining any user-defined functions. The program accepts user input for the number of terms and generates the Fibonacci sequence using a single for loop.

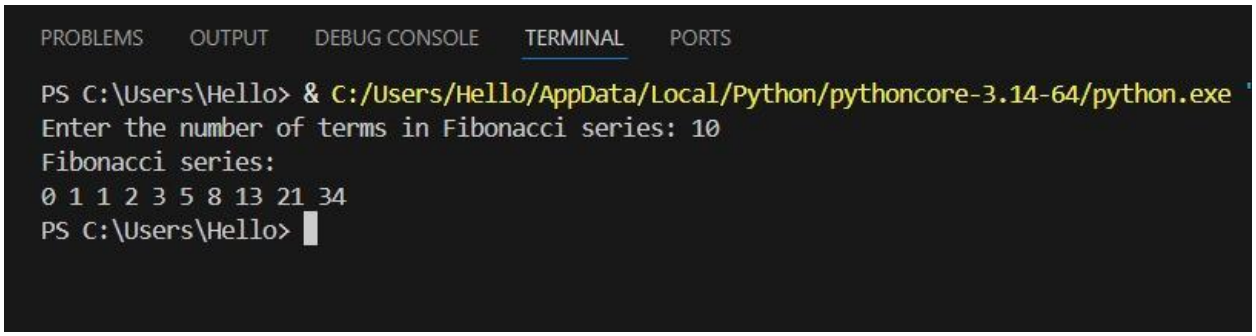
Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

PROMPT
Optimize version of Fibonacci series up to n terms without using functions

CODE:



Output:

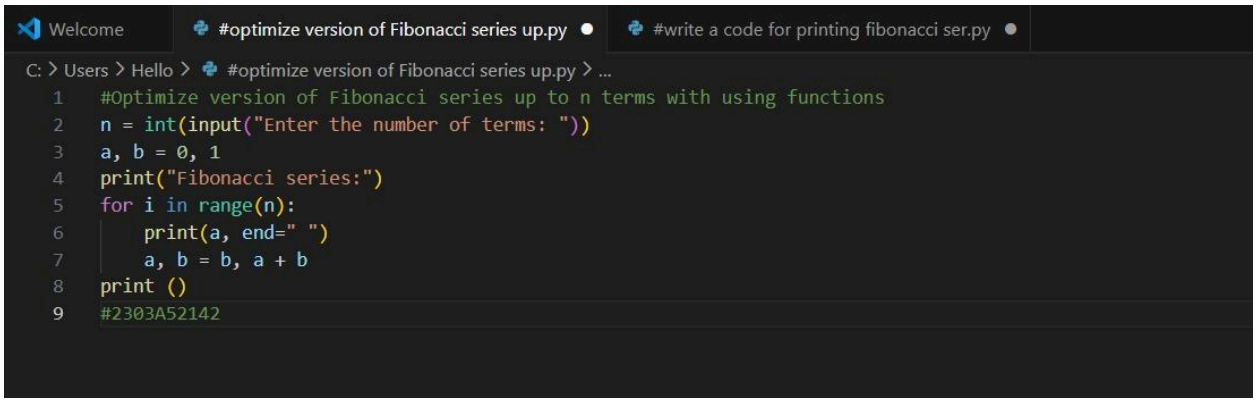


Task 3:Modular Design Using AI Assistance (Fibonacci Using Functions)

PROMPT:

Optimize version of Fibonacci series up to n terms with using functions

CODE:



Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Hello> & C:/Users/Hello/AppData/Local/Python/pythoncore-3.14
Enter the number of terms: 10
Fibonacci series:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\Hello>
```

JUSTIFICATION:
By completing this task, we implemented the Fibonacci series using a user-defined function. The function contains all the Fibonacci logic, which makes the code modular and easy to reuse. Using modularization helps to organize the code properly and allows the same function to be used in different programs. This method makes the code easier to read, easier to debug, and more suitable for large applications.

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

PROMPTS:
Procedural: Write a code for printing fibonacci series up to n terms without using functions
Modular: Optimized version of Fibonacci series up to n terms using functions

CODE:
procedural

Welcome | #write a code for printing fibonacci series without using functions.py

C: > Users > Hello > #write a code for printing fibonacci series without using functions.py > ...

```python
#write a code for printing fibonacci series up to n terms without using functions
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
print("Fibonacci series:")
for i in range(n):
    print(a, end=" ")
    a, b = b, a + b
print ()
# 2303A52142
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Hello> & C:/Users/Hello/AppData/Local/Python/pythoncore-3.14-64/python.exe '
Enter the number of terms in Fibonacci series: 10
Fibonacci series:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\Hello>
```

Modular

```
C: > Users > Hello > #Optimized version of Fibonacci series Modular.py > ...
1   #Optimized version of Fibonacci series up to n terms using functions
2   def fibonacci(n):
3       a, b = 0, 1
4       for i in range(n):
5           print(a, end=" ")
6           a, b = b, a + b
7       print()
8
9   n = int(input("Enter the number of terms: "))
10  fibonacci(n)
11  #2303A52142
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Hello> & C:/Users/Hello/AppData/Local/Python/pythoncore-3.14-64/python.exe
Enter the number of terms: 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\Hello>
```

JUSTIFICATION:

By this task , we are able to find the difference between Procedural(without using functions) and Modular(with using functions). The main use of function is

- Reusability of the code
- Easy to Debug
- Code Clarity
- Suitable for large systems

By observing, we can analyze that using modular method is a better and clean approach.

Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches for Fibonacci Series)

PROMPTS:
Iterative approach: write a code fibonacci series up to n terms using iterative approach
Recursive approach : write a code fibonacci series up to n terms using recursive approach

CODE:

C: > Users > Hello > #write a code fibonacci series up to n t.py > ...

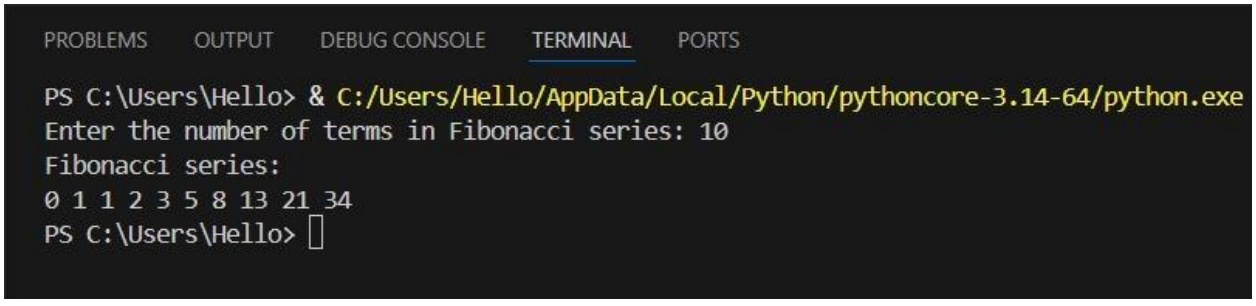
```python
#write a code fibonacci series up to n terms using iteration approach
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
print("Fibonacci series:")
for i in range(n):
    print(a, end=" ")
    a, b = b, a + b
print ()
# 2303A52142
```

C: > Users > Hello > #write a code fibnacci series up to n te.py > ...

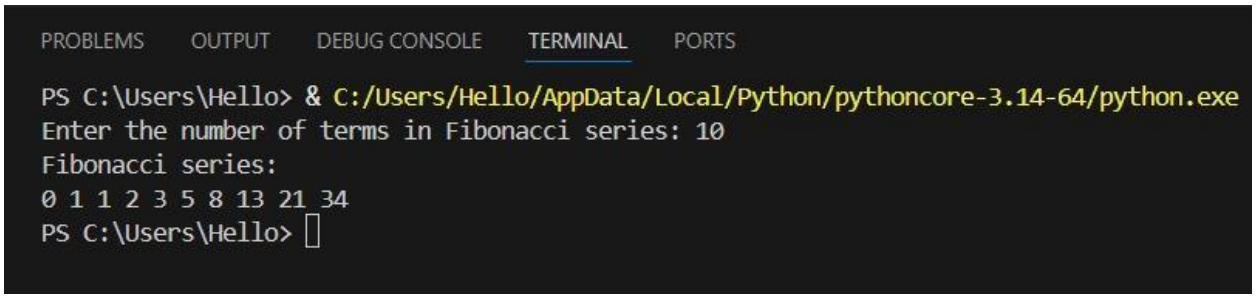
```python
#write a code fibnacci series up to n terms using recursive approach
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

n = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series:")
for i in range(n):
    print(fibonacci(i), end=" ")
print()
# End of the code
# 2303A52142
```

Iterative approach



Recursive approach



Justification:
Using a loop (iterative method) to find Fibonacci numbers is usually better because it's faster, uses less memory, and can handle big numbers easily. The recursive method (using functions that call themselves) is slower, uses more memory, and can crash if the number is too large. That's why in real programs, loops are preferred over recursion.